PERANCANGAN DAN IMPLEMENTASI SISTEM OTOMASI PRIMING DAN STARTING PADA PLTMH TIPE SIPHON HEAD RENDAH

SKRIPSI

ALIF NAYAKA MAHARDIKA

2203411015

PROGRAM STUDI TEKNIK OTOMASI LISTRIK INDUSTRI

JURUSAN TEKNIK ELEKTRO

POLITEKNIK NEGERI JAKARTA

2026

PERANCANGAN DAN IMPLEMENTASI SISTEM OTOMASI PRIMING DAN STARTING PADA PLTMH TIPE SIPHON HEAD RENDAH

SKRIPSI

Diajukan sebagai salah satu syarat untuk memperoleh gelar Sarjana Terapan

ALIF NAYAKA MAHARDIKA

2203411015

PROGRAM STUDI TEKNIK OTOMASI LISTRIK INDUSTRI

JURUSAN TEKNIK ELEKTRO

POLITEKNIK NEGERI JAKARTA

2026

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri dan semua sumber baik yang dikutip maupun dirujuk telah saya nyatakan dengan benar.

Nama	:	ALIF NAYAKA MAHARDIKA
NIM	:	2203411015
Tanda Tangan	:	
Tanggal	:	2 Juli 2026

LEMBAR PENGESAHAN

SKRIPSI

Tugas Akhir diajukan oleh:

Nama : Alif Nayaka Mahardika

NIM : 2203411015

Program Studi : Teknik Otomasi Listrik Industri

Judul Tugas Akhir : Perancangan dan Implementasi Sistem Otomasi *Priming* dan *Starting* pada PLTMH Tipe *Siphon Head* Rendah.

Telah diuji oleh tim penguji dalam Sidang Tugas Akhir pada hari Kamis, 2 Juli 2026 dan dinyatakan **LULUS.**

Pembimbing I : Arum Kusuma Wardhany, S.T., M.T.
(NIP. 199107132020122013) (.....)

Pembimbing II : Nagib Muhammad, S.T., M.T.
(NIP. 199406052022031007) (.....)

Depok, 21 Juli 2026

Disahkan oleh

Ketua Jurusan Teknik Elektro

Dr. Murie Dwiyaniti, S. T., M. T.

(NIP. 197803312003122002)

KATA PENGANTAR

Puji dan syukur saya panjatkan ke hadirat Tuhan Yang Maha Esa, karena atas rahmat dan karunia-Nya penulis dapat menyelesaikan skripsi yang berjudul *"Perancangan dan Implementasi Sistem Otomasi Priming dan Starting pada PLTMH Tipe Siphon Head Rendah"* dengan baik.

Skripsi ini disusun sebagai salah satu syarat untuk menyelesaikan pendidikan pada Program Studi Teknik Otomasi Listrik Industri di Politeknik Negeri Jakarta. Dalam proses penyusunan skripsi ini, penulis mendapatkan banyak bantuan, bimbingan, serta dukungan dari berbagai pihak. Oleh karena itu, pada kesempatan ini penulis ingin menyampaikan ucapan terima kasih kepada:

1. Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya;
2. Ibu Arum Kusuma Wardhany, S.T., M.T. dan Bapak Nagib Muhammad, S.T, M.T., selaku dosen pembimbing yang telah menyediakan waktu, tenaga, dan pikiran untuk mengarahkan penulis dalam penyusunan skripsi ini;
3. Orang tua dan keluarga yang telah memberikan doa serta dukungan.
4. Seluruh dosen dan staf di Politeknik Negeri Jakarta;
5. Rekan kelompok tugas skripsi dan sahabat yang telah banyak membantu penulis dalam menyelesaikan skripsi ini.

Akhir kata, penulis berharap skripsi ini dapat memberikan manfaat bagi pembaca serta menjadi referensi dalam pengembangan sistem PLTMH khususnya tipe *siphon* dengan *head* rendah.

Depok, 2 Juli 2026

Alif Nayaka Mahardika

Perancangan dan Implementasi Sistem Otomasi *Priming* dan *Starting* pada PLTMH Tipe *Siphon Head* Rendah

ABSTRAK

Pembangkit Listrik Tenaga Mikro Hidro (PLTMH) merupakan salah satu solusi pemanfaatan energi terbarukan yang ramah lingkungan dan berpotensi besar untuk dikembangkan, terutama pada daerah dengan sumber air melimpah. Pada kondisi *head* rendah, sistem PLTMH tipe *siphon* menjadi alternatif yang efektif karena tidak memerlukan bendungan besar. Namun, proses *priming* dan *starting* pada sistem *siphon* umumnya masih dilakukan secara manual sehingga kurang efisien dan berpotensi menimbulkan kegagalan operasi. Penelitian ini bertujuan untuk merancang dan mengimplementasikan sistem otomasi *priming* dan *starting* pada PLTMH tipe *siphon head* rendah. Sistem yang dikembangkan menggunakan mikrokontroler ESP32 sebagai pengendali utama, *vacuum* motor sebagai alat bantu proses *priming*, serta *motorized valve* sebagai aktuator untuk mengatur aliran udara pada sistem *vacuum*. Selain itu, sistem dilengkapi dengan *liquid separator tank* yang berfungsi sebagai pemisah antara air dan udara *vacuum* guna melindungi *vacuum* motor. Sistem ini juga dilengkapi dengan sensor water level serta *tachometer* untuk memantau kondisi operasi. Metode penelitian meliputi perancangan sistem, pembuatan prototipe, serta pengujian kinerja sistem. Parameter yang dianalisis meliputi waktu *priming*, kestabilan *siphon*, serta keandalan sistem otomasi. Hasil pengujian menunjukkan bahwa sensor *tachometer* memiliki rata-rata *error* sebesar 1,35% dibandingkan alat ukur *optical tachometer*. Sistem mampu melakukan *starting* otomatis dengan waktu rata-rata 39,1 detik hingga generator mencapai putaran operasi normal. Pengujian penghentian sistem menunjukkan waktu rata-rata 13,38 detik hingga generator berhenti berputar. Fitur auto *restart* berhasil mengembalikan sistem ke kondisi operasi setelah terjadi kehilangan putaran generator. Pada pengujian operasi kontinu, sistem mampu beroperasi hingga 5 jam tanpa terjadinya putus *siphon* dengan putaran generator berada pada rentang 513–522 RPM. Daya listrik tertinggi yang berhasil dicapai sebesar 85,4 W dengan daya rata-rata selama pengujian sekitar 63 W. Berdasarkan hasil tersebut, sistem otomasi yang dirancang mampu menjalankan proses *priming, starting*, penghentian sistem, dan auto *restart* secara otomatis sehingga dapat meningkatkan keandalan operasi PLTMH tipe *siphon head* rendah serta mengurangi ketergantungan terhadap operator.

Kata Kunci : *PLTMH, siphon, priming, otomasi, mikrokontroler* ESP32*, liquid separator tank, tachometer, auto restart.*

Design and Implementation of Priming and Starting Automation System on Low Head Siphon Type Micro Hydro Power Plant

ABSTRACT

Micro Hydro Power Plants (MHP) are one of the environmentally friendly renewable energy solutions with great potential for development, especially in areas with abundant water resources. In low head conditions, the siphon-type MHP system is an effective alternative because it does not require a large dam. However, the priming and starting processes in the siphon system are generally still carried out manually, making it less efficient and potentially causing operational failures. This study aims to design and implement an automated priming and starting system for a low-head siphon-type MHP. The developed system uses an ESP32 microcontroller as the main controller, a vacuum motor as a priming process aid, and a motorized valve as an actuator to regulate air flow in the vacuum system. In addition, the system is equipped with a liquid separator tank that functions as a separator between air and vacuum air to protect the vacuum motor. This system is also equipped with a water level sensor and a tachometer to integrate operating conditions. The research method includes system design, prototype creation, and system performance testing. The parameters described include priming time, siphon stability, and automation system consistency. The test results show that the tachometer sensor has an average error of 1.35% compared to the optical tachometer. The system is capable of performing an automatic start with an average time of 39.1 seconds until the generator reaches normal operating speed. The system freeze test showed an average time of 13.38 seconds until the generator stops rotating. The auto restart feature successfully returns the system to operating conditions after a loss of generator rotation. In continuous operation testing, the system was able to operate for up to 5 hours without a siphon break with generator rotation in the range of 513–522 RPM. The highest electrical power achieved was 85.4 W with an average power during the test of around 63 W. Based on these results, the designed automation system is capable of carrying out the priming, starting, stopping the system, and auto restart processes automatically, thereby improving the operation of the low-head siphon type MHP and reducing dependence on operators.

***Keywords**: PLTMH, siphon, priming, otomasi, mikrokontroler* ESP32*, liquid separator tank, tachometer, auto restart.*

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR RUMUS

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1 Latar Belakang

Kebutuhan energi listrik yang terus meningkat menuntut pemanfaatan sumber energi yang dapat diperbarui dan berkelanjutan. Salah satu sumber energi terbarukan yang memiliki potensi besar untuk dikembangkan khususnya di Indonesia adalah energi air. Selain pembangkit listrik tenaga air skala besar, pemanfaatan energi air dalam skala kecil yang biasa disebut mikrohidro dan pikohidro juga semakin banyak diterapkan karena mampu menyediakan energi listrik dengan teknologi yang relatif sederhana dan biaya operasional yang rendah (Choifin, 2024; Faisal, 2025).

Pembangkit Listrik Tenaga Mikrohidro (PLTMH) merupakan sistem pembangkit yang memanfaatkan energi potensial dan energi kinetik air untuk menghasilkan energi listrik melalui turbin dan generator. Besarnya daya yang dapat dihasilkan dipengaruhi oleh beberapa faktor, di antaranya debit aliran air dan tinggi jatuh air (*head*). Pada lokasi yang memiliki debit air cukup besar namun *head* yang rendah, pemanfaatan energi air tetap dapat dilakukan dengan menggunakan teknologi pembangkit yang dirancang khusus untuk kondisi *head* rendah (*low-head hydropower)* (Gunadi, 2022; Sinagra, 2022).

Perkembangan Pembangkit Listrik Tenaga Air *head* rendah menunjukkan bahwa sumber daya air yang sebelumnya dianggap kurang ekonomis masih memiliki potensi untuk dimanfaatkan sebagai sumber energi listrik. Berbagai penelitian mengenai turbin *low-head* menunjukkan bahwa energi air dapat dimanfaatkan pada saluran irigasi, sungai kecil, maupun saluran air buatan dengan perbedaan elevasi yang relatif rendah (Quaranta, 2020; Shamsuddeen, 2024). Oleh karena itu, diperlukan metode yang mampu meningkatkan pemanfaatan energi air pada kondisi *head* rendah tanpa memerlukan pembangunan infrastruktur yang kompleks.

Salah satu metode yang dapat diterapkan adalah sistem *siphon*. Sistem ini memanfaatkan perbedaan tekanan dan gaya gravitasi untuk mengalirkan air melalui pipa tertutup setelah udara di dalam pipa berhasil dikeluarkan. Penggunaan sistem

siphon memiliki beberapa keunggulan, seperti konstruksi yang sederhana, kemudahan instalasi, dan kemampuan untuk diterapkan pada lokasi yang memiliki perbedaan elevasi terbatas. Penelitian mengenai penerapan *siphon* pada fasilitas air menunjukkan bahwa metode ini dapat digunakan sebagai alternatif dalam pemanfaatan energi air pada lokasi dengan kondisi *head* rendah (Qin, 2019; Mamadalievich, 2023).

Meskipun memiliki beberapa keunggulan, sistem *siphon* memiliki tantangan utama pada proses *priming* dan *starting*. Proses *priming* dilakukan untuk mengeluarkan udara yang berada di dalam pipa sehingga terbentuk kolom air yang kontinu. Apabila masih terdapat udara yang terjebak di dalam pipa, maka aliran yang terbentuk menjadi tidak stabil dan dapat menurunkan kinerja sistem. Beberapa penelitian menunjukkan bahwa proses pembentukan vakum dan proses *self-priming* merupakan faktor yang sangat menentukan keberhasilan pembentukan aliran pada sistem *siphon* (Shu, 2024; Zhang, 2025).

Selain proses *priming*, proses *starting* juga menjadi faktor penting dalam pengoperasian sistem. Penelitian mengenai karakteristik transien pada sistem pompa *siphon* menunjukkan bahwa tahap awal pembentukan aliran memiliki pengaruh yang besar terhadap kestabilan operasi sistem secara keseluruhan (Xiaowen Zhang, 2021). Oleh karena itu, diperlukan suatu metode pengendalian yang mampu mengatur tahapan *priming* dan *starting* secara berurutan dan konsisten.

Perkembangan teknologi mikrokontroler memungkinkan proses tersebut dilakukan secara otomatis melalui integrasi sensor dan aktuator. Pada penelitian ini digunakan mikrokontroler ESP32 sebagai pengendali utama yang menerima masukan dari sensor *float water level* dan sensor *tachometer*, kemudian mengendalikan motor vakum, *motorized valve vacuum*, *motorized valve vent*, serta indikator sistem. Melalui penerapan sistem otomasi diharapkan proses pembentukan *siphon* dapat berlangsung lebih cepat, lebih andal, dan mampu mengurangi ketergantungan terhadap operator.

1.1 Perumusan Masalah

Dari latar belakang tersebut, terdapat beberapa rumusan masalah pada penelitian ini adalah sebagai berikut:

1. Bagaimana merancang sistem otomasi *priming* dan *starting* pada PLTMH tipe *siphon head rendah* berbasis ESP32?
2. Bagaimana mengintegrasikan sensor dan aktuator yang digunakan dalam sistem otomasi yang dirancang?
3. Bagaimana mekanisme kerja sistem untuk melakukan proses *priming*, *starting*, dan *re-starting* secara otomatis?
4. Bagaimana kinerja sistem yang dirancang berdasarkan parameter waktu *priming*, keberhasilan *starting*, dan putaran generator?

1.2 Tujuan Penelitian

Berdasarkan rumusan masalah yang telah dijabarkan, terdapat tujuan dari dilakukannya pada penelitian ini, adapun berbagai tujuannya yaitu:

1. Merancang sistem otomasi *priming* dan *starting* pada PLTMH tipe *siphon head rendah* berbasis ESP32.
2. Mengimplementasikan integrasi sensor dan aktuator ke dalam sistem kontrol otomatis.
3. Mengembangkan logika kontrol yang mampu menjalankan proses *priming*, *starting*, *re-starting*, dan penghentian operasi sistem secara otomatis.
4. Menguji kinerja sistem yang telah dirancang berdasarkan parameter pengujian yang telah ditentukan.

1.3 Luaran

Luaran yang dihasilkan dari penulisan penelitian ini adalah:

1 *Prototype* sistem otomasi *priming* dan *starting* pada PLTMH tipe *siphon head rendah* berbasis ESP32.

2 Laporan tugas akhir yang memuat hasil perancangan, implementasi, dan pengujian sistem yang telah dikembangkan.

3 Artikel ilmiah yang disusun berdasarkan hasil penelitian dan diharapkan dapat dipublikasikan pada seminar atau jurnal ilmiah yang relevan.

4 Media pembelajaran berupa *prototype* dan dokumentasi teknis yang dapat digunakan untuk mendukung kegiatan praktikum dan penelitian di Program Studi D4 Teknik Otomasi Listrik Industri.

5 Kontribusi dalam bidang IPTEKS melalui penerapan sistem otomasi pada PLTMH tipe *siphon head rendah* untuk meningkatkan kemudahan dan

keandalan pengoperasian.

6 Kontribusi bagi masyarakat melalui pengembangan teknologi energi terbarukan yang berpotensi dimanfaatkan pada sumber air dengan *head* rendah

BAB V
PENUTUP

Berdasarkan hasil perancangan, realisasi, dan pengujian sistem otomasi *priming* dan *starting* PLTMH tipe *siphon head* rendah, dapat diambil beberapa kesimpulan sebagai berikut:

1. Sistem otomasi *priming* dan *starting* PLTMH tipe *siphon head* rendah berbasis ESP32 berhasil direalisasikan dan mampu mengendalikan motor vakum, *motorized ball valve* (MV) vacuum, *motorized ball valve* vent, dan *solenoid valve* drain sesuai dengan logika program yang telah dirancang.
2. Sistem pembacaan RPM menggunakan sensor Hall yang terintegrasi pada generator BLDC mampu bekerja dengan baik dengan rata-rata kesalahan pengukuran sebesar 1,35% dibandingkan hasil pengukuran menggunakan *optical tachometer*.
3. Sistem berhasil melakukan proses *starting* otomatis pada seluruh percobaan dengan tingkat keberhasilan 100% dan rata-rata waktu *start-up* sebesar 39,10 detik hingga mencapai kondisi *RUN*.
4. Sistem mampu menghentikan operasi pembangkit secara otomatis melalui pembukaan jalur *vent* dengan rata-rata waktu penghentian sebesar 13,38 detik sejak tombol *Stop* ditekan hingga generator berhenti berputar.
5. Fitur *auto restart* berhasil bekerja sesuai rancangan dengan mendeteksi penurunan RPM generator dan menjalankan kembali proses *starting* secara otomatis hingga sistem kembali mencapai kondisi *RUN*.
6. Fungsi proteksi kegagalan *starting* berhasil bekerja dengan mengaktifkan kondisi *Error* apabila generator tidak mampu mencapai putaran minimum 450 RPM dalam waktu 2 menit.
7. Fungsi proteksi gangguan berulang setelah *auto restart* berhasil bekerja dengan mengaktifkan kondisi *Error* apabila gangguan kembali terjadi dalam periode pemantauan satu menit setelah sistem berhasil melakukan *auto restart.*
8. Sistem berhasil beroperasi secara kontinu hingga 5 jam tanpa mengalami putus *siphon* dengan putaran generator yang relatif stabil pada rentang 513–522 RPM.

Pengujian dihentikan secara sengaja oleh peneliti sehingga batas maksimum durasi operasi kontinu sistem belum dapat ditentukan pada penelitian ini.

9. Sistem PLTMH tipe *siphon head* rendah yang dibuat ini mampu menghasilkan daya keluaran dengan nilai puncak sebesar 85,4 W dan daya rata-rata sekitar 63 W secara kontinu selama pengujian berlangsung pada musim kemarau.

Terdapat beberapa hal yang disarankan sebagai pengembangan guna meningkatkan kinerja sistem di masa mendatang yaitu:

1. Penelitian selanjutnya dapat menambahkan sensor tekanan atau sensor vakum untuk memantau kondisi vakum pada pipa *siphon* secara langsung.
2. Pengujian lebih lanjut dapat dilakukan pada berbagai variasi debit air dan tinggi *head* untuk mengetahui pengaruhnya terhadap waktu *starting*, kestabilan operasi, dan efisiensi sistem.
3. Penelitian selanjutnya dapat mengembangkan desain turbin yang lebih efisien sesuai karakteristik debit dan *head* sistem. Hal ini dikarenakan turbin yang digunakan pada penelitian ini merupakan turbin yang dibeli jadi dan belum dirancang secara khusus untuk kondisi operasi PLTMH tipe *siphon*.

DAFTAR PUSTAKA

Chen, Y., Feng, J., Zhu, D. Z., Xu, H., & Qian, S. (2023). Air pocket removal during siphon priming. Journal of Hydraulic Engineering, 149(7), 04023017.

Choifin, M., Putra, A. P. S., & Afifah, Y. N. (2024). Pengujian alat pembangkit listrik tenaga pico hydro (PLTPH) terhadap keluaran daya dan debit air yang dihasilkan dengan variasi ketinggian head. Mechonversio: Mechanical Engineering Journal, 7(1), 6–14.

Faisal, A., Anisah, S., & Tama, I. A. (2025). Studi perbandingan output daya aktual dan teoritis pada PLTMH. Impression: Jurnal Teknologi dan Informasi, 4(3), 483–493.

Gunadi, G. G. R., Widiawaty, C. D., Utomo, M. P., Satria, R. A., Abimanyu, M. R., Syuriadi, A., & Kamal, D. M. (2022). Pengembangan model pembangkit listrik tenaga picohydro tipe turbin cross flow head rendah. Infotekmesin, 13(2), 201–205.

Hercog, D., Lerher, T., Truntič, M., & Težak, O. (2023). Design and implementation of ESP32-based IoT devices. Sensors, 23(15), 6739.

Junior, A. B., Soares, H. V., Freitas, R. L., de Mesquita, R. N., da Silva Rocha, M., & de Andrade, D. A. (2024). Improving the RELAP5 code modeling of the siphon break effect in a pool type research reactor.

Lee, K. Y., & Kim, W. S. (2017). Study of siphon breaker experiment and simulation for a research reactor. Journal of Visualized Experiments: JOVE, (127), 55972.

Mamadalievich, M. M. (2023). Application of micro-hydroelectric power stations in water barriers using siphon pipes in water facilities. Central Asian Journal of Theoretical and Applied Science, 4(12), 218–223.

Microchip Technology Inc. (2002). Brushless DC motor control made easy (Application Note AN857). Chandler: Microchip Technology Inc.

Qin, L., Leon, A. S., Bian, L. L., Dong, L. L., Verma, V., & Yolcu, A. (2019). A remotely-operated siphon system for water release from wetlands and shallow ponds. IEEE Access, 7, 157680–157687.

Quaranta, E., Bonjean, M., Cuvato, D., Nicolet, C., Dreyer, M., Gaspoz, A., ... & Bragato, N. (2020). Hydropower case study collection: Innovative low head and ecologically improved turbines, hydropower in existing infrastructures, hydropeaking reduction, digitalization and governing systems. Sustainability, 12(21), 8873.

Sabila, M. F. (2025). Desain programmable logic controller (PLC) berbasis

mikrokontroler ESP32 (Doctoral dissertation, Universitas Gadjah Mada).

Scheffler, M. (2021). Low-cost hydropower turbines for developing countries.

Shamsuddeen, M. M., Shahzer, M. A., Roh, M. S., & Kim, J. H. (2024). Feasibility study of ultra-low-head hydro turbines for energy extraction from shallow waterways. Heliyon, 10(15).

Shomayramov, M., Norkulov, B., Rakhmanov, J., Tadjiyeva, D., & Suyunov, J. (2019). Experimental researches of hydraulic vacuum breakdown devices of siphon outlets of pumping stations. In E3S Web of Conferences (Vol. 97, p. 05009). EDP Sciences.

Shu, J., Wang, J., Chen, K., Shen, Q., & Sun, H. (2024). Analysis of factors affecting vacuum formation and drainage in the siphon-vacuum drainage method for marine reclamation. Journal of Marine Science and Engineering, 12(3), 430.

Sinagra, M., Picone, C., Picone, P., Aricò, C., Tucciarelli, T., & Ramos, H. M. (2022). Low-head hydropower for energy recovery in wastewater systems. Water, 14, 1649.

Zhang, X., Tang, F., Liu, C., Shi, L., Liu, H., Sun, Z., & Hu, W. (2021). Numerical simulation of transient characteristics of start-up transition process of large vertical siphon axial flow pump station. Frontiers in Energy Research, 9, 706975.

Zhang, Y. L., Huang, H. F., Zhang, K. Y., & Zheng, S. H. (2025). The design of external-mixing self-priming pump and visualized experiment during self-priming process. International Journal of Energy Research, 2025(1), 4880195.

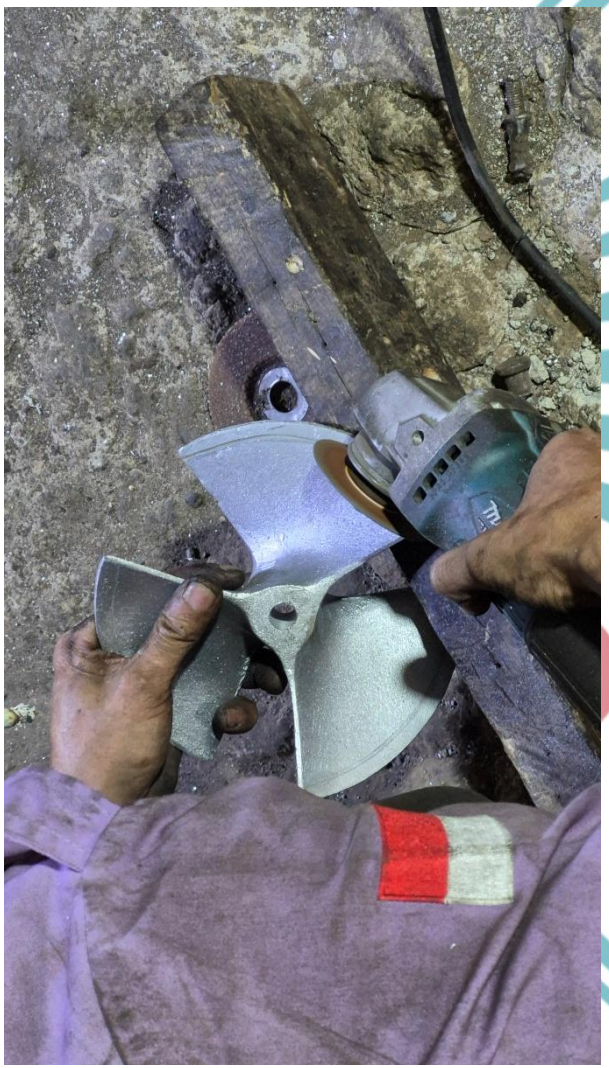
DAFTAR RIWAYAT HIDUP PENULIS

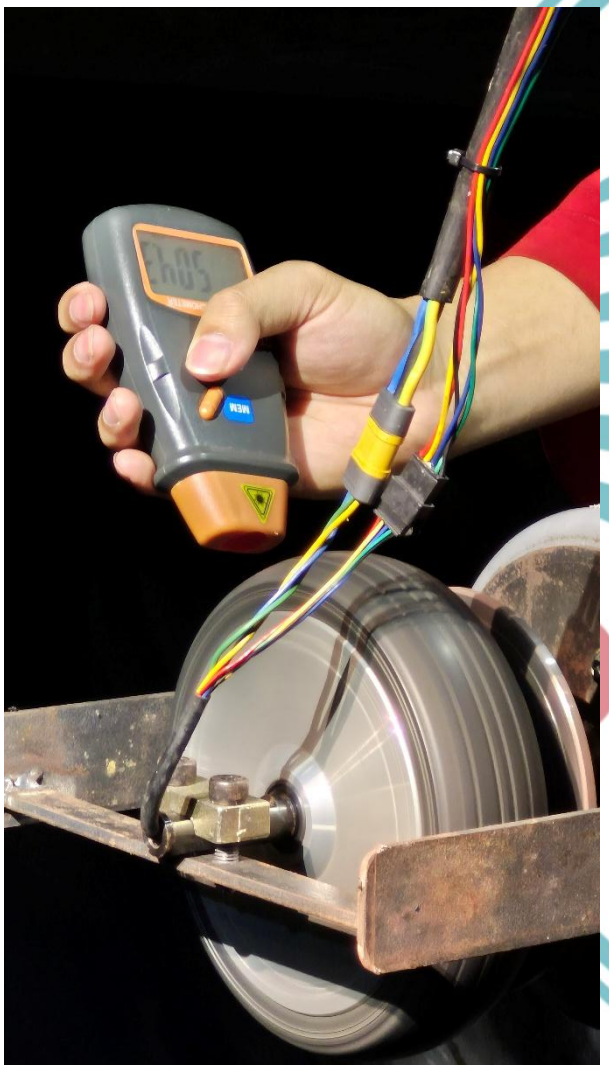
Alif Nayaka Mahardika

Lulus dari SDN Pondok Kacang Barat 03 tahun 2016, SMPN 14 Tangerang Selatan tahun 2019, dan SMA Muhammadiyah 2 Cipondoh tahun 2022. Penulis menjalani pendidikan lanjut di Politeknik Negeri Jakarta dengan Program Studi D4 Teknik Otomasi Listrik Industri, Jurusan Teknik Elektro, Politeknik Negeri Jakarta.

LAMPIRAN

Lampiran 1 Dokumentasi Pengerjaan





Lampiran 2 Program ESP32

```
#include <ModbusMaster.h>                // Library Modbus RTU untuk membaca PZEM-017.

const uint8_t PIN_TACH = 27;               // Input tachometer 15 pulse per revolution.
const uint8_t PIN_FLOAT_WATER = 39;        // Input float, LOW berarti air terdeteksi.
const uint8_t PIN_SEL_AUTO = 34;           // Input selector Auto, LOW berarti Auto dipilih.
const uint8_t PIN_SEL_MANUAL = 35;         // Input selector Manual, LOW berarti Manual dipilih.
const uint8_t PIN_BTN_START = 32;          // Input tombol Start, LOW saat ditekan.
const uint8_t PIN_BTN_STOP = 33;           // Input tombol Stop, LOW saat ditekan.
const uint8_t PIN_BTN_RESET = 25;          // Input tombol Reset, LOW saat ditekan.
const uint8_t PIN_BTN_MOTOR_VAC = 26;      // Input tombol toggle Motor Vacuum manual.
const uint8_t PIN_BTN_MV_VAC = 13;         // Input tombol toggle MV Vacuum manual.
const uint8_t PIN_BTN_MV_VENT = 4;         // Input tombol toggle MV Vent manual.
const uint8_t PIN_BTN_SLV_DRAIN = 18;      // Input tombol toggle SLV Drain manual.

const uint8_t PIN_OUT_MOTOR_VAC = 2;       // Output Motor Vacuum, HIGH berarti ON.
const uint8_t PIN_OUT_MV_VAC = 5;          // Output MV Vacuum open, HIGH berarti ON.
const uint8_t PIN_OUT_MV_VENT = 14;        // Output MV Vent open, HIGH berarti ON.
const uint8_t PIN_OUT_LAMP_READY = 19;     // Output pilot lamp Ready, HIGH berarti ON.
const uint8_t PIN_OUT_LAMP_STARTING = 21;  // Output pilot lamp Starting, HIGH berarti ON.
const uint8_t PIN_OUT_LAMP_RUN = 22;       // Output pilot lamp Run, HIGH berarti ON.
const uint8_t PIN_OUT_LAMP_ERROR = 23;     // Output pilot lamp Error, HIGH berarti ON.
const uint8_t PIN_OUT_SLV_DRAIN = 15;      // Output SLV Drain open, HIGH berarti ON.

const uint8_t PIN_PZEM_RX = 16;            // RX2 ESP32 untuk menerima data dari MAX485.
const uint8_t PIN_PZEM_TX = 17;            // TX2 ESP32 untuk mengirim data ke MAX485.
const int8_t PIN_RS485_DE_RE = 12;         // Pin tambahan DE dan RE MAX485 jika modul tidak auto-direction.
const bool USE_RS485_DIR_PIN = true;       // Ubah false jika MAX485 sudah auto-direction.

const uint8_t PZEM_SLAVE_ID = 1;           // Alamat Modbus bawaan PZEM-017.
const uint32_t PZEM_BAUDRATE = 9600;       // Baudrate Modbus PZEM-017.
const uint16_t TACH_PULSE_PER_REV = 15;    // Jumlah pulse tachometer tiap satu putaran.
const float RPM_NORMAL_MIN = 450.0;        // Batas RPM normal untuk masuk running.
const float RPM_LOCK_MIN = 400.0;          // Batas RPM untuk lock Auto dan proteksi tombol.
const float RPM_ZERO_LIMIT = 1.0;          // Batas RPM yang dianggap nol.
const uint32_t TACH_SAMPLE_MS = 1000;      // Waktu sampling RPM.
const uint32_t TACH_MIN_PULSE_US = 500;    // Filter noise pulse tachometer.
const uint32_t DEBOUNCE_MS = 40;           // Waktu debounce tombol.

const uint32_t AUTO_MV_TO_MOTOR_MS = 4000;  // Delay 4 detik sebelum Motor Vacuum ON.
const uint32_t AUTO_START_TIMEOUT_MS = 110000; // Timeout 1 menit untuk start normal mencapai 450 RPM.
const uint32_t AUTO_RESTART_TIMEOUT_MS = 60000; // Timeout 1 menit untuk auto restart mencapai 450 RPM.
const uint32_t AUTO_AFTER_450_DELAY_MS = 8000; // Delay 8 detik setelah RPM mencapai 450.
const uint32_t AUTO_VENT_PULSE_MS = 14000;  // Lama MV Vent ON untuk venting timer.
const uint32_t AUTO_RESTART_MONITOR_MS = 60000; // Batas 1 menit setelah auto restart running.
const uint32_t ERROR_BLINK_FAST_MS = 150;   // Interval kedip cepat error.
const uint32_t ERROR_BLINK_SLOW_MS = 700;   // Interval kedip lambat error.
const uint32_t PZEM_READ_MS = 1000;         // Interval baca PZEM.
const uint32_t SERIAL_SEND_MS = 1000;         // Interval kirim serial.

enum SelectorMode { MODE_OFF, MODE_AUTO, MODE_MANUAL }; // Daftar mode selector.

enum AutoState {                          // Daftar tahap mode Auto.
  AUTO_WAIT_WATER,                        // Menunggu air terdeteksi.
  AUTO_WAIT_START,                        // Ready dan menunggu tombol Start.
  AUTO_START_MOTOR_DELAY,                 // MV Vacuum ON dan menunggu 4 detik.
  AUTO_START_WAIT_RPM,                    // Motor Vacuum ON dan menunggu 450 RPM.
  AUTO_AFTER_450_DELAY,                   // MV Vacuum OFF dan menunggu 8 detik.
  AUTO_RUNNING,                           // Running normal.
  AUTO_STOP_VENTING,                      // Stop saat running lalu MV Vent ON sampai RPM nol.
  AUTO_EXTERNAL_RUNNING,                  // RPM 450 sudah ada sebelum Start.
  AUTO_EXTERNAL_STOP_VENTING              // Stop dari external running.
};                                        // Akhir daftar tahap Auto.

const char *modeName(SelectorMode mode) {     // Fungsi mengubah enum mode menjadi teks untuk MQTT/debug.
  if (mode == MODE_AUTO) {                    // Mengecek apakah mode Auto.
    return "auto";                            // Mengembalikan teks auto.
  }                                           // Akhir mode Auto.
  if (mode == MODE_MANUAL) {                  // Mengecek apakah mode Manual.
    return "manual";                          // Mengembalikan teks manual.
  }                                           // Akhir mode Manual.
  return "off";                               // Mengembalikan teks off untuk kondisi lain.
}
```

```
72
73  const char *autoStateName(AutoState state) {  // Fungsi mengubah tahapan Auto menjadi teks.
74    switch (state) {                            // Memilih teks berdasarkan nilai AutoState.
75      case AUTO_WAIT_WATER: return "wait_water"; // Teks untuk menunggu air.
76      case AUTO_WAIT_START: return "ready_wait_start"; // Teks untuk ready menunggu start.
77      case AUTO_START_MOTOR_DELAY: return "preopen_vacuum"; // Teks untuk MV Vacuum pre-open.
78      case AUTO_START_WAIT_RPM: return "wait_rpm";    // Teks untuk menunggu RPM.
79      case AUTO_AFTER_450_DELAY: return "close_vacuum_delay"; // Teks untuk delay setelah vacuum ditutup.
80      case AUTO_RUNNING: return "running";      // Teks untuk running.
81      case AUTO_STOP_VENTING: return "stopping_vent"; // Teks untuk stop dengan vent.
82      case AUTO_EXTERNAL_RUNNING: return "ex_run"; // Teks untuk RPM 450 sudah ada sebelum Start.
83      case AUTO_EXTERNAL_STOP_VENTING : return "ex_stop_vent"; // Teks untuk Stop dari external running.
84    }                                           // Akhir switch AutoState.
85    return "unknown";                           // Teks cadangan bila state tidak dikenal.
86  }
87
88  enum TimedVentAction {                    // Daftar aksi setelah MV Vent 14 detik selesai.
89    VENT_NONE,                              // Tidak ada timer MV Vent.
90    VENT_RETURN_READY,                      // Kembali ke ready setelah vent.
91    VENT_AUTO_RESTART,                      // Mulai auto restart setelah vent.
92    VENT_ERROR_FAST,                        // Tetap error kedip cepat setelah vent.
93    VENT_ERROR_SLOW,                        // Tetap error kedip lambat setelah vent.
94    VENT_OFF_MODE                           // Tetap mode Off setelah vent.
95  };                                        // Akhir daftar aksi vent.
96
97  struct PzemData {                         // Struktur data hasil baca PZEM.
98    bool valid;                             // True jika pembacaan PZEM berhasil.
99    float voltage;                          // Tegangan dalam Volt.
100   float current;                          // Arus dalam Ampere.
101   float power;                            // Daya dalam Watt.
102   float energyWh;                         // Energi dalam Wh.
103 };                                        // Akhir struktur data PZEM.
104
105 class Button {                            // Kelas tombol dengan debounce sederhana.
106 public:                                   // Bagian public agar bisa dipakai program utama.
107   uint8_t pin;                            // Nomor pin tombol.
108   bool stablePressed;                     // Status stabil tombol.
109   bool lastRawPressed;                    // Status mentah pembacaan sebelumnya.
110   bool pressedEvent;                      // Event sekali saat tombol baru ditekan.
111   uint32_t lastChangeMs;                  // Waktu perubahan terakhir untuk debounce.
112
113   void begin(uint8_t pinNumber) {         // Fungsi inisialisasi tombol.
114     pin = pinNumber;                      // Menyimpan nomor pin tombol.
115     stablePressed = false;                // Mengawali tombol sebagai tidak ditekan.
116     lastRawPressed = false;               // Mengawali pembacaan mentah sebagai tidak ditekan.
117     pressedEvent = false;                 // Mengosongkan event tombol.
118     lastChangeMs = millis();              // Menyimpan waktu awal debounce.
119     pinMode(pin, INPUT_PULLUP);           // Mengaktifkan pull-up internal untuk tombol aktif LOW.
120   }                                       // Akhir fungsi inisialisasi tombol.
121
122   void update() {                         // Fungsi memperbarui status tombol.
123     bool rawPressed = (digitalRead(pin) == LOW); // Membaca tombol, LOW berarti ditekan.
124     if (rawPressed != lastRawPressed) {   // Mengecek apakah pembacaan mentah berubah.
125       lastRawPressed = rawPressed;        // Menyimpan pembacaan mentah terbaru.
126       lastChangeMs = millis();            // Menyimpan waktu perubahan.
127     }                                     // Akhir pengecekan perubahan mentah.
128     if ((millis() - lastChangeMs) >= DEBOUNCE_MS && rawPressed != stablePressed) { // Mengecek status stabil.
129       stablePressed = rawPressed;         // Menyimpan status stabil terbaru.
130       if (stablePressed) {                // Mengecek apakah tombol baru stabil ditekan.
131         pressedEvent = true;              // Membuat event tekan satu kali.
132       }                                   // Akhir pengecekan tombol ditekan.
133     }                                     // Akhir pengecekan debounce.
134   }                                       // Akhir fungsi update tombol.
135
136   bool wasPressed() {                     // Fungsi membaca event tombol sekali.
137     bool result = pressedEvent;           // Menyalin event tombol.
138     pressedEvent = false;                 // Menghapus event agar tidak terbaca berulang.
139     return result;                        // Mengembalikan hasil event.
140   }                                       // Akhir fungsi event tombol.
141 };                                        // Akhir kelas Button.
142
143 HardwareSerial PzemSerial(2);             // UART2 ESP32 untuk MAX485.
144 ModbusMaster pzemNode;                    // Objek Modbus master.
```

```
Button btnStart;                        // Objek tombol Start.
Button btnStop;                         // Objek tombol Stop.
Button btnReset;                        // Objek tombol Reset.
Button btnMotorVac;                     // Objek tombol Motor Vacuum manual.
Button btnMvVac;                        // Objek tombol MV Vacuum manual.
Button btnMvVent;                       // Objek tombol MV Vent manual.
Button btnSlvDrain;                     // Objek tombol SLV Drain manual.

volatile uint32_t tachPulseCount = 0;   // Jumlah pulse tachometer dari interrupt.
volatile uint32_t lastTachPulseUs = 0;  // Waktu pulse tachometer terakhir.
float currentRpm = 0.0;                 // RPM terbaru.
uint32_t lastRpmSampleMs = 0;           // Waktu sampling RPM terakhir.
SelectorMode activeMode = MODE_OFF;     // Mode yang sedang dijalankan.
AutoState autoState = AUTO_WAIT_WATER;  // Tahap Auto yang sedang dijalankan.
uint32_t autoStateStartMs = 0;          // Waktu mulai tahap Auto.
TimedVentAction timedVentAction = VENT_NONE; // Aksi setelah timer MV Vent selesai.
bool timedVentActive = false;           // True jika MV Vent timer 14 detik sedang aktif.
uint32_t timedVentStartMs = 0;          // Waktu mulai MV Vent timer.
bool outMotorVac = false;               // Status output Motor Vacuum.
bool outMvVac = false;                  // Status output MV Vacuum.
bool outMvVent = false;                 // Status output MV Vent.
bool outLampReady = false;              // Status lampu Ready.
bool outLampStarting = false;           // Status lampu Starting.
bool outLampRun = false;                // Status lampu Run.
bool outLampError = false;              // Status lampu Error.
bool outSlvDrain = false;               // Status output SLV Drain.
bool manualMotorVac = false;            // Toggle manual Motor Vacuum.
bool manualMvVac = false;               // Toggle manual MV Vacuum.
bool manualMvVent = false;              // Toggle manual MV Vent.
bool manualSlvDrain = false;            // Toggle manual SLV Drain.
bool autoRestartCycle = false;          // True jika start berasal dari auto restart.
bool monitorRestartRun = false;         // True jika running hasil restart sedang dipantau 1 menit.
uint32_t restartRunStartMs = 0;         // Waktu mulai running setelah restart.
bool errorLatched = false;              // True jika error terkunci sampai Reset.
bool errorFastBlink = true;             // True untuk kedip cepat, false untuk kedip lambat.

PzemData pzemData = { false, 0, 0, 0, 0 }; // Data PZEM terakhir.
uint32_t lastPzemReadMs = 0;            // Waktu baca PZEM terakhir.
uint32_t lastSerialSendMs = 0;            // Waktu kirim serial terakhir.

void IRAM_ATTR onTachPulse() {          // Interrupt setiap pulse tachometer.
  uint32_t nowUs = micros();            // Membaca waktu sekarang dalam mikrodetik.
  if ((nowUs - lastTachPulseUs) >= TACH_MIN_PULSE_US) { // Mengabaikan noise yang terlalu cepat.
    tachPulseCount++;                   // Menambah jumlah pulse.
    lastTachPulseUs = nowUs;            // Menyimpan waktu pulse terakhir.
  }                                     // Akhir filter pulse.
}                                       // Akhir interrupt tachometer.

void preTransmission() {                // Fungsi sebelum Modbus transmit.
  if (USE_RS485_DIR_PIN) {              // Mengecek apakah pin arah RS485 dipakai.
    digitalWrite(PIN_RS485_DE_RE, HIGH); // Mengaktifkan transmit MAX485.
  }                                     // Akhir pengecekan pin arah.
}                                       // Akhir fungsi sebelum transmit.

void postTransmission() {               // Fungsi setelah Modbus transmit.
  if (USE_RS485_DIR_PIN) {              // Mengecek apakah pin arah RS485 dipakai.
    digitalWrite(PIN_RS485_DE_RE, LOW);  // Mengaktifkan receive MAX485.
  }                                     // Akhir pengecekan pin arah.
}                                       // Akhir fungsi setelah transmit.

void setup() {                          // Setup berjalan sekali saat ESP32 menyala.
  Serial.begin(115200);                 // Mengaktifkan Serial Monitor.
  pinMode(PIN_TACH, INPUT_PULLUP);      // Mengatur tachometer sebagai input pull-up.
  pinMode(PIN_FLOAT_WATER, INPUT);      // Mengatur float sebagai input eksternal pull-up.
  pinMode(PIN_SEL_AUTO, INPUT);         // Mengatur selector Auto sebagai input eksternal pull-up.
  pinMode(PIN_SEL_MANUAL, INPUT);       // Mengatur selector Manual sebagai input eksternal pull-up.
  pinMode(PIN_OUT_MOTOR_VAC, OUTPUT);   // Mengatur Motor Vacuum sebagai output.
  pinMode(PIN_OUT_MV_VAC, OUTPUT);      // Mengatur MV Vacuum sebagai output.
  pinMode(PIN_OUT_MV_VENT, OUTPUT);     // Mengatur MV Vent sebagai output.
  pinMode(PIN_OUT_LAMP_READY, OUTPUT);  // Mengatur lampu Ready sebagai output.
  pinMode(PIN_OUT_LAMP_STARTING, OUTPUT);// Mengatur lampu Starting sebagai output.
  pinMode(PIN_OUT_LAMP_RUN, OUTPUT);    // Mengatur lampu Run sebagai output.
```

```
  pinMode(PIN_OUT_LAMP_ERROR, OUTPUT);   // Mengatur lampu Error sebagai output.
  pinMode(PIN_OUT_SLV_DRAIN, OUTPUT);    // Mengatur SLV Drain sebagai output.
  if (USE_RS485_DIR_PIN) {               // Mengecek apakah pin arah RS485 dipakai.
    pinMode(PIN_RS485_DE_RE, OUTPUT);    // Mengatur DE/RE MAX485 sebagai output.
    digitalWrite(PIN_RS485_DE_RE, LOW);  // Mengawali MAX485 pada mode receive.
  }                                      // Akhir pengaturan pin RS485.
  btnStart.begin(PIN_BTN_START);         // Mengatur tombol Start.
  btnStop.begin(PIN_BTN_STOP);           // Mengatur tombol Stop.
  btnReset.begin(PIN_BTN_RESET);         // Mengatur tombol Reset.
  btnMotorVac.begin(PIN_BTN_MOTOR_VAC);  // Mengatur tombol Motor Vacuum manual.
  btnMvVac.begin(PIN_BTN_MV_VAC);        // Mengatur tombol MV Vacuum manual.
  btnMvVent.begin(PIN_BTN_MV_VENT);      // Mengatur tombol MV Vent manual.
  btnSlvDrain.begin(PIN_BTN_SLV_DRAIN);  // Mengatur tombol SLV Drain manual.
  PzemSerial.begin(PZEM_BAUDRATE, SERIAL_8N2, PIN_PZEM_RX, PIN_PZEM_TX); // Mengaktifkan UART2 untuk PZEM 8N2.
  pzemNode.begin(PZEM_SLAVE_ID, PzemSerial); // Mengatur alamat slave PZEM.
  pzemNode.preTransmission(preTransmission); // Memasang kontrol transmit MAX485.
  pzemNode.postTransmission(postTransmission); // Memasang kontrol receive MAX485.
  attachInterrupt(digitalPinToInterrupt(PIN_TACH), onTachPulse, FALLING); // Memasang interrupt tachometer.
  setAllOutputsOff();                    // Memastikan semua output OFF.
  applyOutputs();                        // Mengirim status OFF ke pin fisik.
  lastRpmSampleMs = millis();            // Menyimpan waktu sampling awal.
  autoStateStartMs = millis();           // Menyimpan waktu state awal.
}                                        // Akhir setup.

void loop() {                            // Loop berjalan terus menerus.
  updateRpm();                           // Menghitung RPM.
  updateButtons();                       // Membaca semua tombol.
  readPzemPeriodically();                // Membaca PZEM berkala.
  sendSerialDataIfNeeded();              // Mengirim data serial berkala.
  handleControlLogic();                  // Menjalankan logika mesin.
  updateTimedVent();                     // Memproses MV Vent timer 14 detik.
  updateErrorBlink();                    // Mengatur kedip error.
  applyOutputs();                        // Mengirim status output ke pin.
}                                        // Akhir loop.

void updateButtons() {                   // Fungsi membaca semua tombol.
  btnStart.update();                     // Membaca tombol Start.
  btnStop.update();                      // Membaca tombol Stop.
  btnReset.update();                     // Membaca tombol Reset.
  btnMotorVac.update();                  // Membaca tombol Motor Vacuum manual.
  btnMvVac.update();                     // Membaca tombol MV Vacuum manual.
  btnMvVent.update();                    // Membaca tombol MV Vent manual.
  btnSlvDrain.update();                  // Membaca tombol SLV Drain manual.
}                                        // Akhir pembacaan tombol.

void updateRpm() {                       // Fungsi menghitung RPM.
  uint32_t nowMs = millis();             // Membaca waktu sekarang.
  uint32_t elapsedMs = nowMs - lastRpmSampleMs; // Menghitung waktu sejak sampling terakhir.
  if (elapsedMs >= TACH_SAMPLE_MS) {     // Mengecek apakah waktu sampling terpenuhi.
    noInterrupts();                      // Mengunci interrupt sebentar.
    uint32_t pulses = tachPulseCount;    // Menyalin jumlah pulse.
    tachPulseCount = 0;                  // Mengosongkan counter pulse.
    interrupts();                        // Membuka interrupt kembali.
    currentRpm = (pulses * 60000.0) / (TACH_PULSE_PER_REV * elapsedMs); // Menghitung RPM.
    lastRpmSampleMs = nowMs;             // Menyimpan waktu sampling terbaru.
  }                                      // Akhir sampling RPM.
}                                        // Akhir fungsi RPM.

SelectorMode readSelectorMode() {        // Fungsi membaca selector.
  bool autoSelected = (digitalRead(PIN_SEL_AUTO) == LOW); // Membaca posisi Auto.
  bool manualSelected = (digitalRead(PIN_SEL_MANUAL) == LOW); // Membaca posisi Manual.
  if (autoSelected && !manualSelected) { // Mengecek Auto valid.
    return MODE_AUTO;                    // Mengembalikan mode Auto.
  }                                      // Akhir Auto valid.
  if (manualSelected && !autoSelected) { // Mengecek Manual valid.
    return MODE_MANUAL;                  // Mengembalikan mode Manual.
  }                                      // Akhir Manual valid.
  return MODE_OFF;                       // Selain itu dianggap Off.
}                                        // Akhir baca selector.

bool isWaterDetected() {                 // Fungsi membaca float water level.
  return digitalRead(PIN_FLOAT_WATER) == LOW; // LOW berarti air terdeteksi.
```

```
}                                       // Akhir baca float.

bool isRpmZero() {                      // Fungsi cek RPM nol.
  return currentRpm <= RPM_ZERO_LIMIT;  // True jika RPM dianggap nol.
}                                       // Akhir cek RPM nol.

bool isLockRpm() {                      // Fungsi cek RPM lock Auto.
  return currentRpm >= RPM_LOCK_MIN;    // True jika RPM minimal 400.
}                                       // Akhir cek RPM lock.

bool isBelowLockRpm() {                 // Fungsi cek RPM di bawah 400.
  return currentRpm < RPM_LOCK_MIN;     // True jika RPM kurang dari 400.
}                                       // Akhir cek RPM bawah lock.

bool isNormalRpm() {                    // Fungsi cek RPM normal.
  return currentRpm >= RPM_NORMAL_MIN;  // True jika RPM minimal 450.
}                                       // Akhir cek RPM normal.

bool elapsedSince(uint32_t startMs, uint32_t durationMs) { // Fungsi cek waktu lewat.
  return (millis() - startMs) >= durationMs; // True jika durasi sudah tercapai.
}                                       // Akhir fungsi cek waktu.

void handleControlLogic() {             // Fungsi utama logika kontrol.
  SelectorMode requestedMode = readSelectorMode(); // Membaca selector fisik.
  SelectorMode desiredMode = requestedMode; // Menyiapkan mode yang akan dijalankan.
  if (activeMode == MODE_AUTO && isLockRpm() && !errorLatched) { // Mengecek kunci Auto saat RPM 400 atau lebih.
    desiredMode = MODE_AUTO;            // Mengunci Auto walau selector dipindah.
  }                                     // Akhir kunci Auto.
  if (timedVentActive && timedVentAction == VENT_OFF_MODE) { // Mengecek timer vent karena pindah Off.
    desiredMode = MODE_OFF;             // Menahan mode Off sampai vent selesai.
  }                                     // Akhir cek timer Off.
  if (errorLatched) {                   // Mengecek apakah error terkunci.
    handleErrorLatched(requestedMode);  // Menjalankan logika error.
    return;                             // Menghentikan logika lain.
  }                                     // Akhir cek error.
  if (timedVentActive) {                // Mengecek apakah MV Vent timer aktif.
    handleTimedVentLock();              // Mengunci tombol selama MV Vent ON.
    return;                             // Menghentikan logika lain.
  }                                     // Akhir cek MV Vent timer.
  if (desiredMode != activeMode) {      // Mengecek perubahan mode.
    enterMode(desiredMode);             // Masuk ke mode baru.
    if (timedVentActive) {              // Mengecek apakah pindah mode memulai vent timer.
      return;                           // Keluar agar vent tidak tertimpa logika mode.
    }                                   // Akhir cek vent setelah pindah mode.
  }                                     // Akhir perubahan mode.
  if (activeMode == MODE_OFF) {         // Mengecek mode Off.
    handleOffMode();                    // Menjalankan Off.
  } else if (activeMode == MODE_MANUAL) { // Mengecek mode Manual.
    handleManualMode();                 // Menjalankan Manual.
  } else if (activeMode == MODE_AUTO) {  // Mengecek mode Auto.
    handleAutoMode();                   // Menjalankan Auto.
  }                                     // Akhir pemilihan mode.
}                                       // Akhir logika kontrol.

void handleTimedVentLock() {            // Fungsi mengunci kontrol saat MV Vent timer aktif.
  clearAllButtonEvents();               // Menghapus semua event tombol.
  setAllOutputsOff();                   // Mematikan output lain.
  outMvVent = true;                     // Menyalakan MV Vent selama timer aktif.
}                                       // Akhir lock timer vent.

void handleErrorLatched(SelectorMode requestedMode) { // Fungsi saat error terkunci.
  activeMode = requestedMode;           // Mengikuti selector untuk menentukan tampilan error/off.
  bool resetPressed = btnReset.wasPressed(); // Membaca tombol Reset.
  clearStartStopEvents();               // Menghapus event Start dan Stop.
  clearManualButtonEvents();            // Menghapus event tombol manual.
  setLoadOutputsOff();                  // Mematikan semua beban.
  outLampReady = false;                 // Mematikan Ready.
  outLampStarting = false;              // Mematikan Starting.
  outLampRun = false;                   // Mematikan Run.
  outLampError = false;                 // Menyiapkan lampu Error untuk dikedipkan.
  if (timedVentActive) {                // Mengecek apakah vent error 14 detik masih aktif.
```

```
    outMvVent = true;                   // MV Vent tetap ON selama timer error.
  }                                     // Akhir cek vent error.
  if (resetPressed) {                   // Mengecek apakah Reset ditekan.
    timedVentActive = false;            // Menghentikan timer vent jika masih aktif.
    timedVentAction = VENT_NONE;        // Menghapus aksi vent.
    errorLatched = false;               // Menghapus error terkunci.
    errorFastBlink = true;              // Mengembalikan tipe kedip default.
    autoRestartCycle = false;           // Menghapus status auto restart.
    monitorRestartRun = false;          // Menghapus pemantauan restart.
    setAllOutputsOff();                 // Mematikan semua output.
    enterMode(requestedMode);           // Masuk ulang sesuai selector.
  }                                     // Akhir Reset.
}                                       // Akhir logika error.

void enterMode(SelectorMode newMode) {  // Fungsi masuk ke mode baru.
  SelectorMode oldMode = activeMode;    // Menyimpan mode sebelumnya.
  activeMode = newMode;                 // Menyimpan mode baru.
  setAllOutputsOff();                   // Mematikan semua output.
  resetManualToggles();                 // Menghapus toggle manual.
  autoRestartCycle = false;             // Membatalkan status auto restart.
  monitorRestartRun = false;            // Membatalkan monitor restart.
  if (oldMode == MODE_AUTO && newMode == MODE_OFF) { // Mengecek pindah dari Auto ke Off.
    startTimedVent(VENT_OFF_MODE);      // MV Vent ON 14 detik sesuai revisi Auto nomor 12.
    return;                             // Keluar dari fungsi masuk mode.
  }                                     // Akhir cek Auto ke Off.
  if (newMode == MODE_AUTO) {           // Mengecek apakah masuk Auto.
    setAutoState(AUTO_WAIT_WATER);      // Auto dimulai dari menunggu air.
  }                                     // Akhir masuk Auto.
}                                       // Akhir masuk mode.

void handleOffMode() {                  // Fungsi mode Off.
  clearAllButtonEvents();               // Menghapus semua event tombol.
  setAllOutputsOff();                   // Memastikan semua output OFF.
  resetManualToggles();                 // Memastikan toggle manual OFF.
}                                       // Akhir mode Off.

void handleManualMode() {               // Fungsi mode Manual.
  btnStart.wasPressed();                // Mengabaikan tombol Start.
  btnStop.wasPressed();                 // Mengabaikan tombol Stop.
  btnReset.wasPressed();                // Mengabaikan tombol Reset.
  outLampReady = false;                 // Mematikan Ready.
  outLampStarting = false;              // Mematikan Starting.
  outLampRun = false;                   // Mematikan Run.
  outLampError = false;                 // Mematikan Error.
  if (btnMotorVac.wasPressed()) {       // Mengecek tombol Motor Vacuum manual.
    manualMotorVac = !manualMotorVac;   // Toggle Motor Vacuum.
  }                                     // Akhir tombol Motor Vacuum.
  if (btnMvVac.wasPressed()) {          // Mengecek tombol MV Vacuum manual.
    manualMvVac = !manualMvVac;         // Toggle MV Vacuum.
  }                                     // Akhir tombol MV Vacuum.
  if (btnMvVent.wasPressed()) {         // Mengecek tombol MV Vent manual.
    manualMvVent = !manualMvVent;       // Toggle MV Vent.
  }                                     // Akhir tombol MV Vent.
  if (btnSlvDrain.wasPressed()) {       // Mengecek tombol SLV Drain manual.
    manualSlvDrain = !manualSlvDrain;   // Toggle SLV Drain.
  }                                     // Akhir tombol SLV Drain.
  outMotorVac = manualMotorVac;         // Output Motor mengikuti toggle.
  outMvVac = manualMvVac;               // Output MV Vacuum mengikuti toggle.
  outMvVent = manualMvVent;             // Output MV Vent mengikuti toggle.
  outSlvDrain = manualSlvDrain;         // Output SLV Drain mengikuti toggle.
}                                       // Akhir mode Manual.

void handleAutoMode() {                 // Fungsi mode Auto.
  clearManualButtonEvents();            // Menghapus tombol manual karena tidak berlaku di Auto.
  if (autoState == AUTO_WAIT_WATER) {   // Mengecek tahap tunggu air.
    handleAutoWaitWater();              // Menjalankan tunggu air.
  } else if (autoState == AUTO_WAIT_START) { // Mengecek tahap tunggu Start.
    handleAutoWaitStart();              // Menjalankan tunggu Start.
  } else if (autoState == AUTO_START_MOTOR_DELAY) { // Mengecek tahap delay motor.
    handleAutoStartMotorDelay();        // Menjalankan delay motor.
  } else if (autoState == AUTO_START_WAIT_RPM) { // Mengecek tahap tunggu RPM.
```

```
    handleAutoStartWaitRpm();             // Menjalankan tunggu RPM.
  } else if (autoState == AUTO_AFTER_450_DELAY) { // Mengecek delay setelah 450.
    handleAutoAfter450Delay();            // Menjalankan delay setelah 450.
  } else if (autoState == AUTO_RUNNING) { // Mengecek tahap running.
    handleAutoRunning();                  // Menjalankan running.
  } else if (autoState == AUTO_STOP_VENTING) { // Mengecek tahap venting Stop.
    handleAutoStopVenting();              // Menjalankan venting Stop.
  } else if (autoState == AUTO_EXTERNAL_RUNNING) { // Mengecek external running.
    handleAutoExternalRunning();          // Menjalankan external running.
  } else if (autoState == AUTO_EXTERNAL_STOP_VENTING) { // Mengecek external venting.
    handleAutoExternalStopVenting();      // Menjalankan external venting.
  }                                       // Akhir pemilihan tahap Auto.
}                                         // Akhir mode Auto.

void handleAutoWaitWater() {              // Tahap Auto menunggu air.
  clearStartStopResetEvents();            // Semua tombol utama diabaikan.
  setLoadOutputsOff();                    // Semua beban OFF.
  outLampReady = false;                   // Ready OFF.
  outLampStarting = false;                // Starting OFF.
  outLampRun = false;                     // Run OFF.
  outLampError = false;                   // Error OFF.
  if (isNormalRpm()) {                    // Mengecek RPM sudah 450 sebelum Start.
    beginExternalRunning();               // Masuk external running.
    return;                               // Keluar dari tahap ini.
  }                                       // Akhir cek external RPM.
  if (isWaterDetected()) {                // Mengecek air terdeteksi.
    outLampReady = true;                  // Ready ON.
    setAutoState(AUTO_WAIT_START);        // Pindah tunggu Start.
  }                                       // Akhir cek air.
}                                         // Akhir tunggu air.

void handleAutoWaitStart() {              // Tahap Auto ready menunggu Start.
  bool startPressed = btnStart.wasPressed(); // Membaca tombol Start.
  btnStop.wasPressed();                   // Stop diabaikan sebelum Start.
  btnReset.wasPressed();                  // Reset diabaikan saat tidak error.
  setLoadOutputsOff();                    // Semua beban OFF.
  outLampReady = isWaterDetected();       // Ready ON hanya jika air ada.
  outLampStarting = false;                // Starting OFF.
  outLampRun = false;                     // Run OFF.
  outLampError = false;                   // Error OFF.
  if (isNormalRpm()) {                    // Mengecek RPM sudah 450 sebelum Start.
    beginExternalRunning();               // Masuk external running.
    return;                               // Keluar dari tahap ini.
  }                                       // Akhir cek external RPM.
  if (!isWaterDetected()) {               // Mengecek air hilang.
    setAutoState(AUTO_WAIT_WATER);        // Kembali tunggu air.
    return;                               // Keluar dari tahap ini.
  }                                       // Akhir cek air hilang.
  if (startPressed) {                     // Mengecek Start ditekan.
    beginAutoStarting(false);             // Memulai starting normal.
  }                                       // Akhir cek Start.
}                                         // Akhir tunggu Start.

void handleAutoStartMotorDelay() {        // Tahap MV Vacuum ON delay 4 detik.
  bool stopPressed = btnStop.wasPressed(); // Membaca tombol Stop.
  btnStart.wasPressed();                  // Start diabaikan saat starting.
  btnReset.wasPressed();                  // Reset diabaikan saat tidak error.
  outLampReady = false;                   // Ready OFF.
  outLampStarting = true;                 // Starting ON.
  outLampRun = false;                     // Run OFF.
  outLampError = false;                   // Error OFF.
  outMvVac = true;                        // MV Vacuum ON.
  outMotorVac = false;                    // Motor Vacuum masih OFF.
  outMvVent = false;                      // MV Vent OFF.
  if (stopPressed && isBelowLockRpm()) {  // Stop berlaku jika RPM masih di bawah 400.
    startTimedVent(VENT_RETURN_READY);    // Semua output OFF lalu MV Vent ON 14 detik.
    return;                               // Keluar dari tahap ini.
  }                                       // Akhir cek Stop.
  if (isNormalRpm()) {                    // Mengecek RPM sudah 450.
    beginAfter450Delay();                 // Masuk delay 8 detik.
    return;                               // Keluar dari tahap ini.
  }                                       // Akhir cek RPM 450.
  if (elapsedSince(autoStateStartMs, AUTO_MV_TO_MOTOR_MS)) { // Mengecek 4 detik selesai.
```

```
503     outMotorVac = true;                   // Motor Vacuum ON.
504     setAutoState(AUTO_START_WAIT_RPM);    // Pindah menunggu RPM.
505   }                                       // Akhir delay 4 detik.
506 }                                         // Akhir delay motor.
507
508 void handleAutoStartWaitRpm() {           // Tahap menunggu RPM mencapai 450.
509   bool stopPressed = btnStop.wasPressed(); // Membaca tombol Stop.
510   btnStart.wasPressed();                  // Start diabaikan saat starting.
511   btnReset.wasPressed();                  // Reset diabaikan saat tidak error.
512   outLampReady = false;                   // Ready OFF.
513   outLampStarting = true;                 // Starting ON.
514   outLampRun = false;                     // Run OFF.
515   outLampError = false;                   // Error OFF.
516   outMvVac = true;                        // MV Vacuum ON.
517   outMotorVac = true;                     // Motor Vacuum ON.
518   outMvVent = false;                      // MV Vent OFF.
519   if (stopPressed && isBelowLockRpm()) {  // Stop berlaku jika RPM masih di bawah 400.
520     startTimedVent(VENT_RETURN_READY);    // Semua output OFF lalu MV Vent ON 14 detik.
521     return;                               // Keluar dari tahap ini.
522   }                                       // Akhir cek Stop.
523   if (isNormalRpm()) {                    // Mengecek RPM mencapai 450.
524     beginAfter450Delay();                 // MV Vacuum OFF lalu delay 8 detik.
525     return;                               // Keluar dari tahap ini.
526   }                                       // Akhir cek RPM 450.
527   if (autoRestartCycle && elapsedSince(autoStateStartMs, AUTO_RESTART_TIMEOUT_MS)) { // Mengecek timeout 1 menit restart.
528     startTimedVent(VENT_ERROR_SLOW);      // Error lambat jika restart gagal mencapai 450.
529     return;                               // Keluar dari tahap ini.
530   }                                       // Akhir cek timeout restart.
531   if (!autoRestartCycle && elapsedSince(autoStateStartMs, AUTO_START_TIMEOUT_MS)) { // Mengecek timeout 2 menit start normal.
532     startTimedVent(VENT_ERROR_FAST);      // Error cepat jika start normal gagal mencapai 450.
533   }                                       // Akhir cek timeout normal.
534 }                                         // Akhir tunggu RPM.
535
536 void handleAutoAfter450Delay() {          // Tahap setelah RPM mencapai 450.
537   clearStartStopResetEvents();            // Semua tombol utama dikunci.
538   outLampReady = false;                   // Ready OFF.
539   outLampStarting = true;                 // Starting tetap ON.
540   outLampRun = false;                     // Run belum ON.
541   outLampError = false;                   // Error OFF.
542   outMvVac = false;                       // MV Vacuum OFF.
543   outMotorVac = true;                     // Motor Vacuum tetap ON.
544   outMvVent = false;                      // MV Vent OFF.
545   if (elapsedSince(autoStateStartMs, AUTO_AFTER_450_DELAY_MS)) { // Mengecek delay 8 detik selesai.
546     outLampStarting = false;              // Starting OFF.
547     outMotorVac = false;                  // Motor Vacuum OFF.
548     outLampRun = true;                    // Run ON.
549     if (autoRestartCycle) {               // Mengecek apakah dari auto restart.
550       monitorRestartRun = true;           // Mengaktifkan monitor 1 menit.
551       restartRunStartMs = millis();       // Menyimpan waktu running restart.
552       autoRestartCycle = false;           // Menghapus status restart aktif.
553     }                                     // Akhir cek auto restart.
554     setAutoState(AUTO_RUNNING);           // Masuk running.
555   }                                       // Akhir delay 8 detik.
556 }                                         // Akhir tahap setelah 450.
557
558 void handleAutoRunning() {                // Tahap running normal.
559   bool stopPressed = btnStop.wasPressed(); // Membaca tombol Stop.
560   btnStart.wasPressed();                  // Start diabaikan saat running.
561   btnReset.wasPressed();                  // Reset diabaikan saat tidak error.
562   outLampReady = false;                   // Ready OFF.
563   outLampStarting = false;                // Starting OFF.
564   outLampRun = true;                      // Run ON.
565   outLampError = false;                   // Error OFF.
566   outMvVac = false;                       // MV Vacuum OFF.
567   outMotorVac = false;                    // Motor Vacuum OFF.
568   if (monitorRestartRun && elapsedSince(restartRunStartMs, AUTO_RESTART_MONITOR_MS)) { // Mengecek 1 menit aman.
569     monitorRestartRun = false;            // Menghapus monitor restart.
570   }                                       // Akhir cek monitor.
571   if (stopPressed && isNormalRpm()) {     // Mengecek Stop saat Run dan RPM minimal 450.
572     outMvVent = true;                     // MV Vent ON.
573     setAutoState(AUTO_STOP_VENTING);      // Menunggu RPM nol.
```

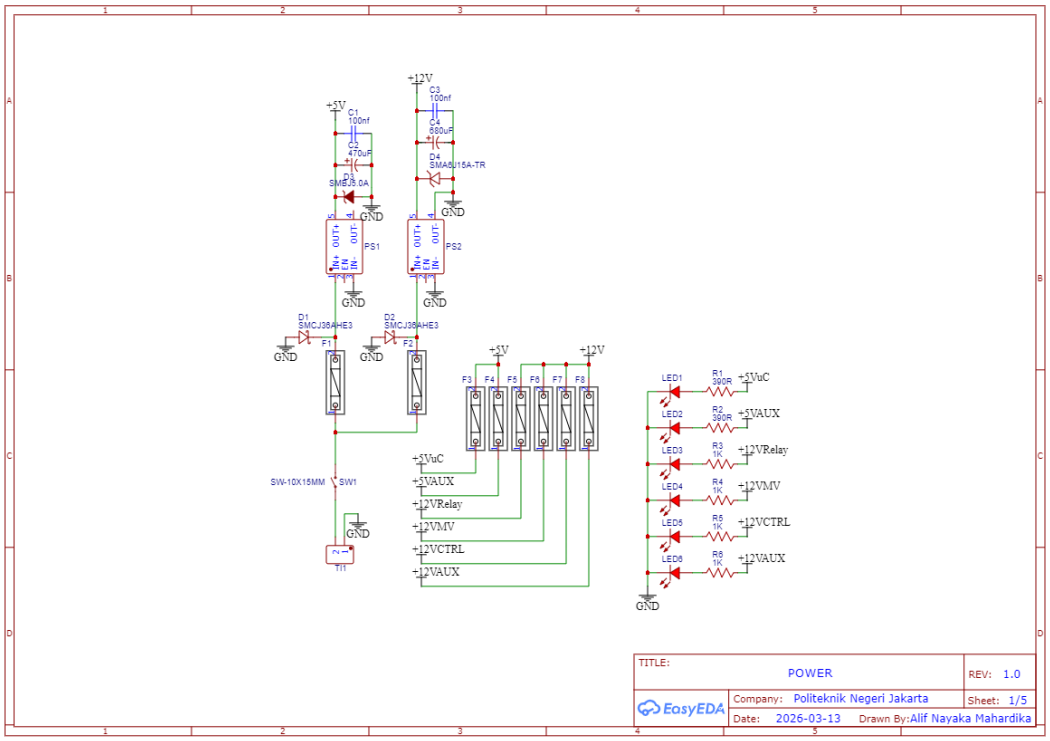
```
574     return;                              // Keluar dari running.
575   }                                      // Akhir cek Stop running.
576   if (isBelowLockRpm()) {                // Mengecek RPM turun di bawah 400.
577     outLampRun = false;                  // Run OFF.
578     if (monitorRestartRun) {             // Mengecek gagal dalam 1 menit setelah restart.
579       startTimedVent(VENT_ERROR_SLOW);   // MV Vent 14 detik lalu error kedip lambat.
580     } else {                             // Jika bukan kegagalan setelah restart.
581       startTimedVent(VENT_AUTO_RESTART); // MV Vent 14 detik lalu auto restart.
582     }                                    // Akhir pilihan RPM turun.
583   }                                      // Akhir cek RPM turun.
584 }                                        // Akhir running.
585
586 void handleAutoStopVenting() {           // Tahap venting setelah Stop running.
587   clearStartStopResetEvents();           // Semua tombol utama dikunci.
588   outLampReady = false;                  // Ready OFF.
589   outLampStarting = false;               // Starting OFF.
590   outLampRun = true;                     // Run ON sampai RPM nol.
591   outLampError = false;                  // Error OFF.
592   outMotorVac = false;                   // Motor Vacuum OFF.
593   outMvVac = false;                      // MV Vacuum OFF.
594   outMvVent = true;                      // MV Vent ON sampai RPM nol.
595   if (isRpmZero()) {                     // Mengecek RPM nol.
596     setAllOutputsOff();                  // Semua output OFF.
597     monitorRestartRun = false;           // Menghapus monitor restart.
598     setAutoState(AUTO_WAIT_WATER);       // Kembali ke awal Auto.
599   }                                      // Akhir cek RPM nol.
600 }                                        // Akhir venting Stop.
601
602 void handleAutoExternalRunning() {       // Tahap RPM 450 sebelum Start.
603   bool stopPressed = btnStop.wasPressed(); // Membaca tombol Stop.
604   btnStart.wasPressed();                 // Start diabaikan.
605   btnReset.wasPressed();                 // Reset diabaikan saat tidak error.
606   setLoadOutputsOff();                   // Beban Auto OFF.
607   outLampReady = false;                  // Ready OFF.
608   outLampStarting = false;               // Starting OFF.
609   outLampRun = isNormalRpm();            // Run ON selama RPM minimal 450.
610   outLampError = false;                  // Error OFF.
611   if (stopPressed && isNormalRpm()) {    // Mengecek Stop saat external running.
612     outMvVent = true;                    // MV Vent ON.
613     setAutoState(AUTO_EXTERNAL_STOP_VENTING); // Masuk external venting.
614     return;                              // Keluar dari tahap ini.
615   }                                      // Akhir cek Stop.
616   if (isRpmZero()) {                     // Mengecek RPM nol.
617     setAllOutputsOff();                  // Semua output OFF.
618     setAutoState(AUTO_WAIT_WATER);       // Kembali tunggu air.
619   }                                      // Akhir cek RPM nol.
620 }                                        // Akhir external running.
621
622 void handleAutoExternalStopVenting() {   // Tahap venting external.
623   clearStartStopResetEvents();           // Semua tombol utama dikunci.
624   outLampReady = false;                  // Ready OFF.
625   outLampStarting = false;               // Starting OFF.
626   outLampRun = true;                     // Run ON sampai RPM nol.
627   outLampError = false;                  // Error OFF.
628   outMotorVac = false;                   // Motor Vacuum OFF.
629   outMvVac = false;                      // MV Vacuum OFF.
630   outMvVent = true;                      // MV Vent ON sampai RPM nol.
631   if (isRpmZero()) {                     // Mengecek RPM nol.
632     setAllOutputsOff();                  // Semua output OFF.
633     setAutoState(AUTO_WAIT_WATER);       // Kembali tunggu air.
634   }                                      // Akhir cek RPM nol.
635 }                                        // Akhir venting external.
636
637 void beginAutoStarting(bool fromRestart) { // Fungsi mulai starting Auto.
638   setAllOutputsOff();                    // Mematikan semua output.
639   outLampStarting = true;                // Starting ON.
640   outMvVac = true;                       // MV Vacuum ON.
641   autoRestartCycle = fromRestart;        // Menyimpan asal start.
642   setAutoState(AUTO_START_MOTOR_DELAY);  // Masuk delay 4 detik.
643 }                                        // Akhir mulai starting.
644
```

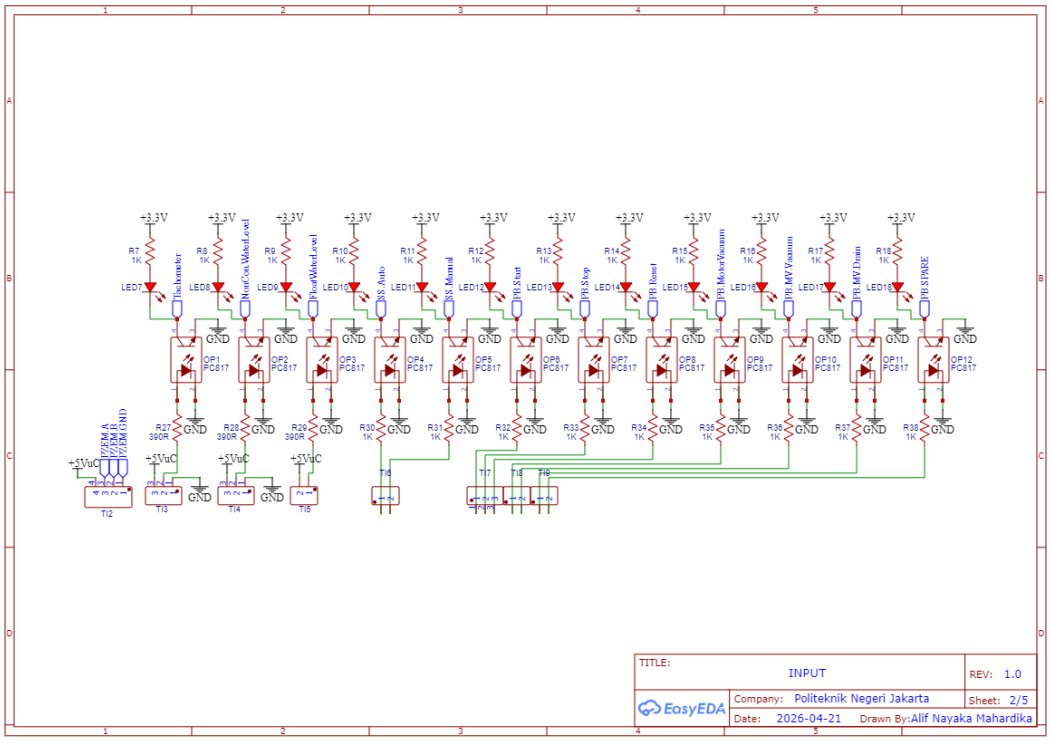
```
645  void beginAfter450Delay() {               // Fungsi saat RPM mencapai 450.
646    outMvVac = false;                       // MV Vacuum OFF.
647    outMotorVac = true;                     // Motor Vacuum tetap ON.
648    outLampStarting = true;                 // Starting tetap ON.
649    setAutoState(AUTO_AFTER_450_DELAY);     // Masuk delay 8 detik.
650  }                                         // Akhir RPM 450.
651
652  void beginExternalRunning() {             // Fungsi masuk external running.
653    setAllOutputsOff();                     // Mematikan output lain.
654    outLampRun = true;                      // Run ON.
655    setAutoState(AUTO_EXTERNAL_RUNNING);    // Masuk external running.
656  }                                         // Akhir external running.
657
658  void startTimedVent(TimedVentAction action) { // Fungsi memulai MV Vent timer 14 detik.
659    setAllOutputsOff();                     // Mematikan semua output lain.
660    outMvVent = true;                       // Menyalakan MV Vent.
661    timedVentActive = true;                 // Menandai timer vent aktif.
662    timedVentAction = action;               // Menyimpan aksi setelah timer.
663    timedVentStartMs = millis();            // Menyimpan waktu mulai timer.
664    if (action == VENT_ERROR_FAST) {        // Mengecek aksi error cepat.
665      errorLatched = true;                  // Mengunci error.
666      errorFastBlink = true;                // Memilih kedip cepat.
667    }                                       // Akhir error cepat.
668    if (action == VENT_ERROR_SLOW) {        // Mengecek aksi error lambat.
669      errorLatched = true;                  // Mengunci error.
670      errorFastBlink = false;               // Memilih kedip lambat.
671    }                                       // Akhir error lambat.
672  }                                         // Akhir mulai timer vent.
673
674  void updateTimedVent() {                  // Fungsi memproses MV Vent timer.
675    if (!timedVentActive) {                 // Mengecek apakah timer tidak aktif.
676      return;                               // Keluar jika tidak ada timer.
677    }                                       // Akhir cek timer aktif.
678    outMvVent = true;                       // Menjaga MV Vent tetap ON.
679    if (elapsedSince(timedVentStartMs, AUTO_VENT_PULSE_MS)) { // Mengecek 14 detik selesai.
680      finishTimedVent();                    // Menyelesaikan timer vent.
681    }                                       // Akhir cek durasi vent.
682  }                                         // Akhir proses timer vent.
683
684  void finishTimedVent() {                  // Fungsi aksi setelah MV Vent 14 detik selesai.
685    TimedVentAction finishedAction = timedVentAction; // Menyalin aksi yang selesai.
686    SelectorMode requestedMode = readSelectorMode(); // Membaca selector setelah vent selesai.
687    timedVentActive = false;                // Mematikan status timer vent.
688    timedVentAction = VENT_NONE;            // Menghapus aksi timer vent.
689    outMvVent = false;                      // Mematikan MV Vent.
690    if (finishedAction == VENT_RETURN_READY) { // Mengecek aksi kembali ready.
691      forceModeAfterVent(requestedMode);    // Mengikuti selector setelah vent selesai.
692    } else if (finishedAction == VENT_AUTO_RESTART) { // Mengecek aksi auto restart.
693      if (requestedMode == MODE_AUTO && isWaterDetected()) { // Mengecek Auto masih dipilih dan air ada.
694        activeMode = MODE_AUTO;             // Memastikan mode tetap Auto.
695        beginAutoStarting(true);            // Memulai auto restart tanpa tombol Start.
696      } else {                              // Jika Auto tidak dipilih atau air tidak ada.
697        forceModeAfterVent(requestedMode);  // Mengikuti selector setelah vent selesai.
698      }                                     // Akhir pilihan auto restart.
699    } else if (finishedAction == VENT_OFF_MODE) { // Mengecek aksi tetap Off.
700      forceModeAfterVent(MODE_OFF);         // Tetap di mode Off.
701    } else if (finishedAction == VENT_ERROR_FAST || finishedAction == VENT_ERROR_SLOW) { // Mengecek aksi error.
702      setAllOutputsOff();                   // Memastikan semua output selain Error OFF.
703    }                                       // Akhir pilihan aksi vent.
704  }                                         // Akhir selesai timer vent.
705
706  void forceModeAfterVent(SelectorMode modeAfterVent) { // Fungsi pindah mode setelah vent tanpa memulai vent baru.
707    activeMode = modeAfterVent;             // Menyimpan mode tujuan.
708    setAllOutputsOff();                     // Mematikan semua output.
709    resetManualToggles();                   // Menghapus toggle manual.
710    autoRestartCycle = false;               // Menghapus status restart.
711    monitorRestartRun = false;              // Menghapus monitor restart.
712    if (modeAfterVent == MODE_AUTO) {       // Mengecek apakah kembali Auto.
713      setAutoState(AUTO_WAIT_WATER);        // Kembali menunggu air.
714    }                                       // Akhir cek Auto.
715  }                                         // Akhir pindah mode setelah vent.
716
```

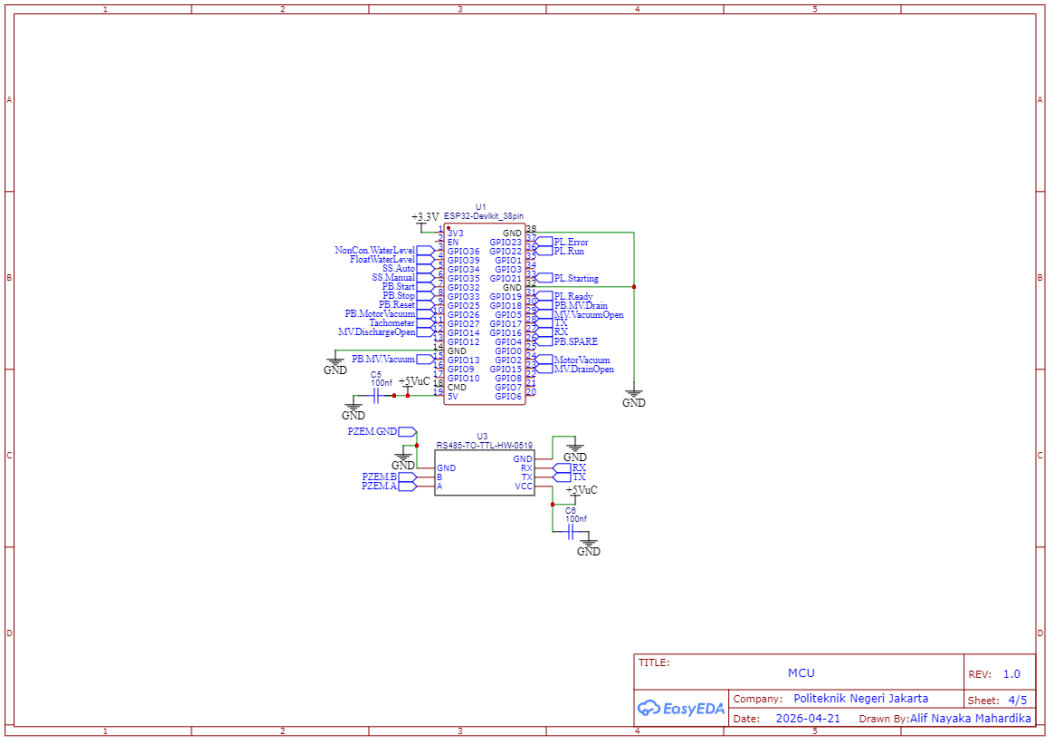
```
717  void setAutoState(AutoState newState) {   // Fungsi mengganti tahap Auto.
718    autoState = newState;                   // Menyimpan tahap baru.
719    autoStateStartMs = millis();            // Menyimpan waktu mulai tahap.
720  }                                         // Akhir ganti tahap.
721
722  void updateErrorBlink() {                 // Fungsi kedip Error.
723    if (!errorLatched) {                    // Mengecek tidak ada error.
724      return;                               // Keluar jika tidak error.
725    }                                       // Akhir cek error.
726    if (activeMode == MODE_OFF) {           // Mengecek selector Off.
727      outLampError = false;                 // Error OFF saat Off.
728      return;                               // Keluar dari kedip.
729    }                                       // Akhir cek Off.
730    uint32_t blinkMs = errorFastBlink ? ERROR_BLINK_FAST_MS : ERROR_BLINK_SLOW_MS; // Memilih interval kedip.
731    outLampError = ((millis() / blinkMs) % 2) == 0; // Membuat pola kedip ON/OFF.
732  }                                         // Akhir kedip Error.
733
734  void setAllOutputsOff() {                 // Fungsi mematikan semua output.
735    outMotorVac = false;                    // Motor Vacuum OFF.
736    outMvVac = false;                       // MV Vacuum OFF.
737    outMvVent = false;                      // MV Vent OFF.
738    outLampReady = false;                   // Ready OFF.
739    outLampStarting = false;                // Starting OFF.
740    outLampRun = false;                     // Run OFF.
741    outLampError = false;                   // Error OFF.
742    outSlvDrain = false;                    // SLV Drain OFF.
743  }                                         // Akhir semua output OFF.
744
745  void setLoadOutputsOff() {                // Fungsi mematikan beban.
746    outMotorVac = false;                    // Motor Vacuum OFF.
747    outMvVac = false;                       // MV Vacuum OFF.
748    outMvVent = false;                      // MV Vent OFF.
749    outSlvDrain = false;                    // SLV Drain OFF.
750  }                                         // Akhir beban OFF.
751
752  void applyOutputs() {                     // Fungsi menulis output fisik.
753    digitalWrite(PIN_OUT_MOTOR_VAC, outMotorVac ? HIGH : LOW); // Menulis Motor Vacuum.
754    digitalWrite(PIN_OUT_MV_VAC, outMvVac ? HIGH : LOW); // Menulis MV Vacuum.
755    digitalWrite(PIN_OUT_MV_VENT, outMvVent ? HIGH : LOW); // Menulis MV Vent.
756    digitalWrite(PIN_OUT_LAMP_READY, outLampReady ? HIGH : LOW); // Menulis Ready.
757    digitalWrite(PIN_OUT_LAMP_STARTING, outLampStarting ? HIGH : LOW); // Menulis Starting.
758    digitalWrite(PIN_OUT_LAMP_RUN, outLampRun ? HIGH : LOW); // Menulis Run.
759    digitalWrite(PIN_OUT_LAMP_ERROR, outLampError ? HIGH : LOW); // Menulis Error.
760    digitalWrite(PIN_OUT_SLV_DRAIN, outSlvDrain ? HIGH : LOW); // Menulis SLV Drain.
761  }                                         // Akhir tulis output.
762
763  void resetManualToggles() {               // Fungsi reset toggle manual.
764    manualMotorVac = false;                 // Toggle Motor Vacuum OFF.
765    manualMvVac = false;                    // Toggle MV Vacuum OFF.
766    manualMvVent = false;                   // Toggle MV Vent OFF.
767    manualSlvDrain = false;                 // Toggle SLV Drain OFF.
768  }                                         // Akhir reset toggle.
769
770  void clearAllButtonEvents() {             // Fungsi hapus semua event tombol.
771    clearStartStopResetEvents();            // Hapus event tombol utama.
772    clearManualButtonEvents();              // Hapus event tombol manual.
773  }                                         // Akhir hapus semua event.
774
775  void clearStartStopEvents() {             // Fungsi hapus event Start dan Stop.
776    btnStart.wasPressed();                  // Hapus event Start.
777    btnStop.wasPressed();                   // Hapus event Stop.
778  }                                         // Akhir hapus Start Stop.
779
780  void clearStartStopResetEvents() {        // Fungsi hapus event Start Stop Reset.
781    btnStart.wasPressed();                  // Hapus event Start.
782    btnStop.wasPressed();                   // Hapus event Stop.
783    btnReset.wasPressed();                  // Hapus event Reset.
784  }                                         // Akhir hapus tombol utama.
785
786  void clearManualButtonEvents() {          // Fungsi hapus event tombol manual.
787    btnMotorVac.wasPressed();               // Hapus event Motor Vacuum manual.
```

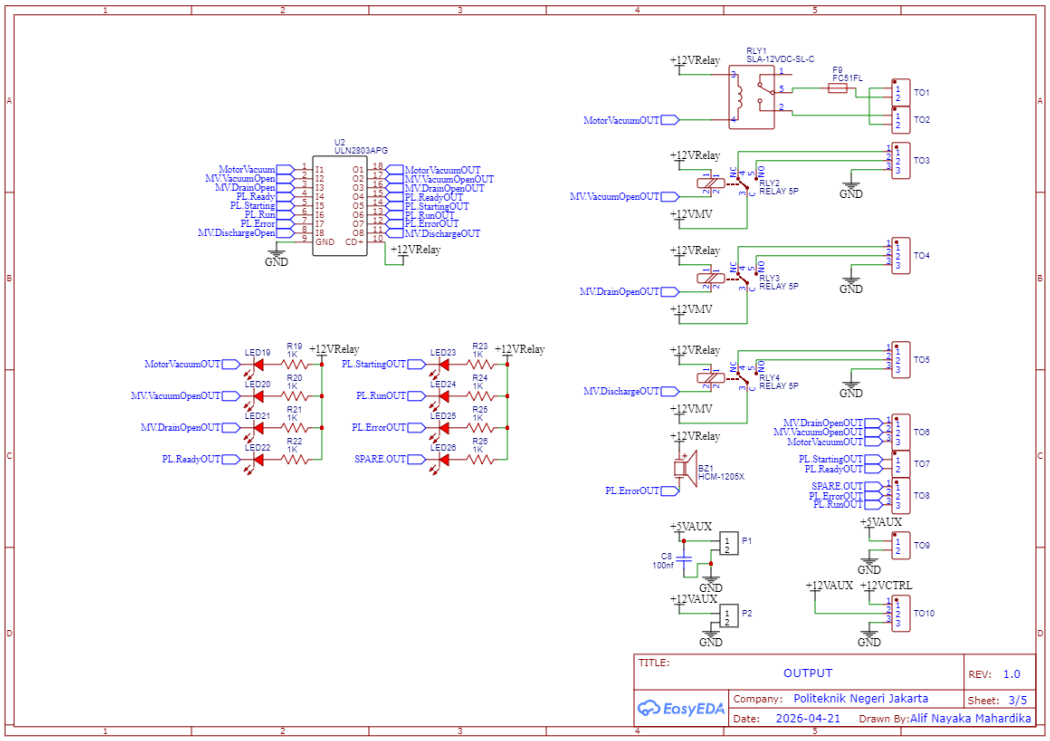
```
788   btnMvVac.wasPressed();                  // Hapus event MV Vacuum manual.
789   btnMvVent.wasPressed();                 // Hapus event MV Vent manual.
790   btnSlvDrain.wasPressed();               // Hapus event SLV Drain manual.
791 }                                         // Akhir hapus tombol manual.
792
793 void readPzemPeriodically() {             // Fungsi baca PZEM berkala.
794   if (!elapsedSince(lastPzemReadMs, PZEM_READ_MS)) { // Mengecek interval baca.
795     return;                               // Keluar jika belum waktunya.
796   }                                       // Akhir cek interval.
797   lastPzemReadMs = millis();              // Menyimpan waktu baca.
798   uint8_t result = pzemNode.readInputRegisters(0x0000, 8); // Membaca 8 input register PZEM.
799   if (result == pzemNode.ku8MBSuccess) {  // Mengecek Modbus berhasil.
800     uint16_t voltageRaw = pzemNode.getResponseBuffer(0); // Raw tegangan.
801     uint16_t currentRaw = pzemNode.getResponseBuffer(1); // Raw arus.
802     uint16_t powerLow = pzemNode.getResponseBuffer(2); // Raw daya low word.
803     uint16_t powerHigh = pzemNode.getResponseBuffer(3); // Raw daya high word.
804     uint16_t energyLow = pzemNode.getResponseBuffer(4); // Raw energi low word.
805     uint16_t energyHigh = pzemNode.getResponseBuffer(5); // Raw energi high word.
806     uint32_t powerRaw = ((uint32_t)powerHigh << 16) | powerLow; // Menggabungkan daya 32 bit.
807     uint32_t energyRaw = ((uint32_t)energyHigh << 16) | energyLow; // Menggabungkan energi 32 bit.
808     pzemData.valid = true;                // Menandai PZEM valid.
809     pzemData.voltage = voltageRaw * 0.01; // Skala tegangan menjadi Volt.
810     pzemData.current = currentRaw * 0.01; // Skala arus menjadi Ampere.
811     pzemData.power = powerRaw * 0.1;      // Skala daya menjadi Watt.
812     pzemData.energyWh = energyRaw * 1.0;  // Skala energi menjadi Wh.
813   } else {                                // Jika pembacaan gagal.
814     pzemData.valid = false;               // Menandai data PZEM tidak valid.
815   }                                       // Akhir hasil baca PZEM.
816 }                                         // Akhir baca PZEM.
817
818 void sendSerialDataIfNeeded() {                  // Menggirimkan data PZEM dan rpm ke Pi dengan Serial USB
819   if (millis() - lastSerialSendMs < SERIAL_SEND_MS) { // Mengecek apakah belum waktunya kirim.
820     return;                                     // Keluar agar data tidak dikirim terlalu sering.
821   }                                             // Akhir pemeriksaan interval.
822
823   lastSerialSendMs = millis();                  // Menyimpan waktu pengiriman terakhir.
824
825   char payload[320];                            // Buffer teks JSON tetap sama.
826   snprintf(payload, sizeof(payload),            // Membuat payload JSON sederhana.
827            "{"
828            "\"rpm\":%.1f,"
829            "\"pzem_ok\":%s,"
830            "\"voltage\":%.2f,"
831            "\"current\":%.2f,"
832            "\"power\":%.1f,"
833            "\"energy_wh\":%.0f,"
834            "\"mode\":\"%s\","
835            "\"auto_state\":\"%s\","
836            "\"motor_vac\":%d,"
837            "\"mv_vac\":%d,"
838            "\"mv_vent\":%d,"
839            "\"lamp_ready\":%d,"
840            "\"lamp_starting\":%d,"
841            "\"lamp_run\":%d,"
842            "\"lamp_error\":%d,"
843            "\"slv_drain\":%d"
844            "}",
845            currentRpm,
846            pzemData.valid ? "true" : "false",
847            pzemData.voltage,
848            pzemData.current,
849            pzemData.power,
850            pzemData.energyWh,
851            modeName(activeMode),
852            autoStateName(autoState),
853            outMotorVac,
854            outMvVac,
855            outMvVent,
856            outLampReady,
857            outLampStarting,
858            outLampRun,
859            outLampError,
860            outSlvDrain);
861
862   // Kirim string JSON langsung lewat kabel USB ke Raspberry Pi
863   Serial.println(payload);                      // Akhir fungsi setelah transmit.
864 }
865
```

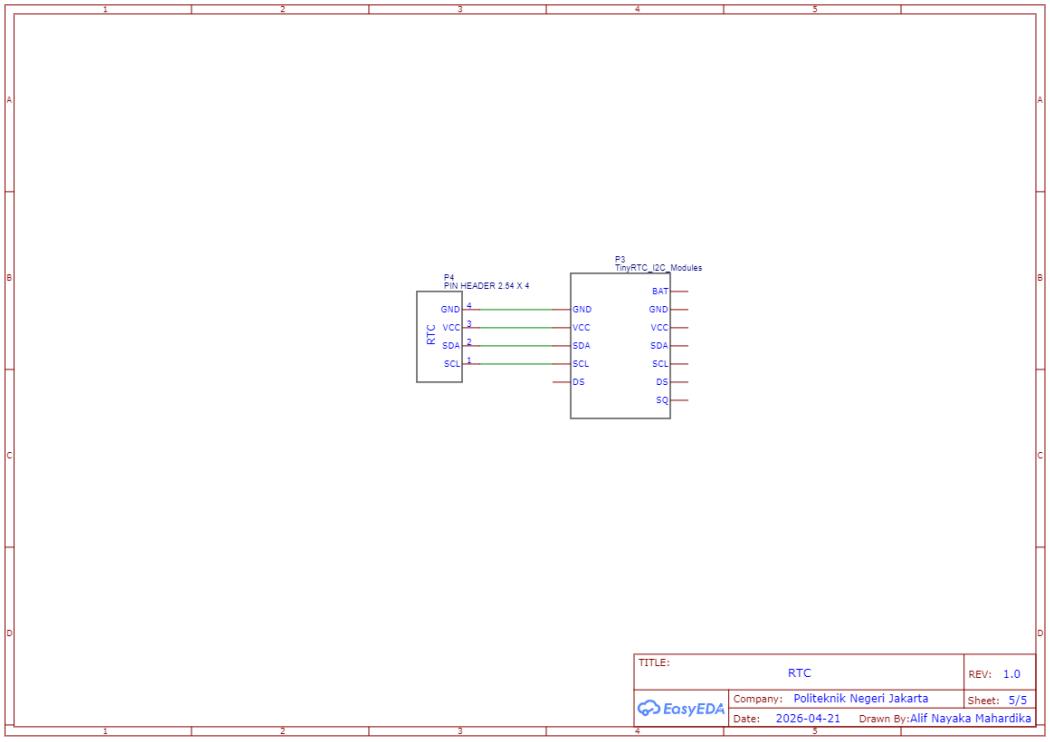
Lampiran 3 Skematik PCB











Lampiran 4 *Wiring Diagram*